7 GARP

7.1 Overview

The module ***PulGARP***, which contains the class ***GARP***, facilitates the use of GARP pulse cycles in GAMMA nmr simulations. Class GARP contains parameters which define how GARP cycles is to be implemented and provides functions for building GARP based waveforms, composite pulses, and pulse trains.

7.2 Chapter Contents

7.2.1 GARP Section Listing

7.2.2 GARP Function Listing

GARP-1 Functions

Access Functions

GARP-1 Functions

Input/Output Functions

7.2.3 GARP Figures & Tables Listing

7.2.4 GARP Examples

7.2.5 GARP Programs

7.3 Constructors and Assignment

7.3.1 GARP

Usage:

```
#include <PulGARP.h>
GARP()
GARP(double gB1, const String& ch, double ph=0, double off=0);
GARP(const GARP& GRP)
```

Description:

The function *GARP* is used to create a GARP parameter container.

1. PulGARP() - Creates an “empty” NULL GARP parameter. Can be later filled by an assignment.
2. PulGARP(double gB1, const& ch, double ph, double off) - Sets up GARP for having an rf-field strength of ***gB1*** Hz on the channel specified by ***ch***. GARP will be applied with an overall phase of ***ph*** degrees and an offset of ***off*** Hz. Called with another PulGARP quantity this function constructs an identical PulGARP to the inputPWF1.
3. PulGARP(const PulGARP &PWF1) - Called with another PulGARP quantity this function constructs an identical PulGARP to the inputPWF1.

Return Value:

GARP returns no parameters. It is used strictly to create a GARP parameter container.

Examples:

```
PulGARP PG;
PulGARP PG1(538.9, “13C”);
PulGARP PG3(PG1);
```

See Also: =

7.3.2 =

Usage:

```
#include <PulGARP.h>
void GARP operator = (PulGARP &PWF1)
```

Description:

The unary operator = (the assignment operator) allows for the setting of one GARP to another GARP. If the GARP being assigned to exists it will be overwritten by the assigned GARP.

Return Value:

None, the function is void

Example:

```
PulGARP PG1(538.9, “13C”);
PulGARP PG3 = PG1;
```

See Also: PulGARP

7.4 Access Functions

7.4.1 channel

Usage:

```
#include <PulGARP.h>
String GARP::channel( )
```

Description:

The function ***channel*** returns a string indicating the isotope channel GARP is applied on.

Return Value:

The function returns a string.

Example:

```
#include <PulGARP.h>
GARP GP(600.0, “13C”);                  // GARP Parameters
cout << “\n\tGARP Decouple On “         // Output channel
     << GP.channel();
```

See Also:

7.4.2 strength

Usage:

```
#include <PulGARP.h>
double GARP::strength()
```

Description:

The function ***strength*** returns the value of the rf-field amplitude used in GARP (in Hz).

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
GARP GP(600.0, “13C”);                  // GARP Parameters
cout << “\n\tGARP Field Strength Is “   // Output rf strength
     << GP.strength() << “ Hz”;         // (will be 600 Hz of course)
```

See Also:

7.4.3 phase

Usage:

```
#include <PulGARP.h>
double GARP::phase()
```

Description:

The function ***phase*** returns the value of the rf-field phase used for GARP in degrees.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
GARP GP;                                    // Declare GARP Parameters
GP.read("filein.pset")                      // Read in GARP Parameters
cout << "\n\tGARP Phase Is "                // Output (overall) rf phase
     << GP.phase() << " degrees";
```

7.4.4 offset

Usage:

```
#include <PulGARP.h>
double GARP::offset()
```

Description:

The function ***offset*** returns the value of the rf-field offset used for GARP in Hz.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
GARP GP;                                    // Declare GARP Parameters
GP.read("filein.pset")                      // Read in GARP Parameters
cout << "\n\tGARP Offset Is "               // Output rf offset
     << GP.offset() << " Hz";
```

7.5 GARP-1 Functions

7.5.1 WF

7.5.2 WF_GARP1

Usage:

```
#include <PulGARP.h>
PulWaveform GARP::WF()
PulWaveform GARP::WF_GARP1()
```

Description:

The ***GARP*** member functions ***WF*** and WF_GARP1 both return the 25 step GARP-1 waveform.

Return Value:

A 25 step GARP-1 pulse waveform is returned.

Example:

```
#include <PulGARP.h>
GARP GP;                              // Declare GARP Parameters
GP.read("filein.pset")                // Read in GARP Parameters
PulWaveform GWF = GP.WF();            // Make GARP waveform (GARP-1)
```

See Also: PCmp, CycGARP1

7.5.3 PCmp

7.5.4 PCmpGARP1

Usage:

```
#include <PulGARP.h>
PulComposite GARP::PCmp(const spin_system& sys)
PulComposite GARP::PCmpGARP1(const spin_system& sys)
```

Description:

The ***GARP*** member functions ***PCmp*** and ***PCmpGARP1*** both return the 25 step GARP-1 composite pulse for the input spin system ***sys***.

Return Value:

A 25 step GARP-1 composite pulse is returned.

Example:

```
spin_system sys;                      // Declare a spin system
sys.read("filein.sys");               // Read in the spin system
```

```
GARP GP;                                  // Declare GARP Parameters
GP.read("filein.pset")                    // Read in GARP Parameters
PulComposite GCP = GP.PCmp(sys);          // GARP composite pulse(GARP-1)
```

See Also: WF, CycGARP1

7.5.5 CycGARP1

Usage:

```
#include <PulGARP.h>
PulCycle GARP::CycGARP1(const spin_system& sys)
```

Description:

The ***GARP*** member function ***CycGARP1*** returns a pulse cycle using the 25 step GARP-1 pulse sequence coupled to a WALTZ-4 cycle.

Return Value:

A GARP-1 pulse cycle is returned.

Example:

```
spin_system sys;                          // Declare a spin system
sys.read("filein.sys");                   // Read in the spin system
GARP GP;                                  // Declare GARP Parameters
GP.read("filein.pset")                    // Read in GARP Parameters
PulCycle GCy = GP.CycGARP1(sys);          // GARP-1 pulse cycle
```

See Also: WF, PCmp

7.6 Propagator Functions

7.6.1 GetU

Usage:

```
#include <PulGARP.h>
gen_op GetU(i)
```

Description:

The function ***GetU*** will return the propagator which is active during step ***i*** of the pulse waveform. The Hamiltonian returned is defined in the rotating frame of the pulse waveform and contains contributions from the pulse waveform rf-field and the isotropic static Hamiltonian.

Return Value:

The function returns an operator.

Example:

```
#include <PulGARP.h>
double tp = 0.01;                    // Set pulse length to 10 ms
int N = 1001;                        // Set number of steps to 1001
```

See Also:

7.7 Input/Output Functions

7.7.1 printBase

Usage:

```
#include <PulGARP.h>
ostr printBase(ostream& ostr)
```

Description:

The function ***printBase*** will put information regarding the GARP parameters into the output stream ***ostr*** given as an input argument. The function will have less embellishment than the similar function ***print***.

Return Value:

The function modifies the output stream and returns it.

Example:

```
GARP GP;
GP.read("filein.pset");
GP.print(cout);
```

See Also: print, <<

7.7.2 print

Usage:

```
#include <PulGARP.h>
ostr print(ostream& ostr)
```

Description:

The function ***print*** will put information regarding the GARP parameters into the output stream ***ostr*** given as an input argument.

Return Value:

The function modifies the output stream and returns it.

Example:

```
GARP GP;
GP.read("filein.pset");
GP.print(cout);
```

See Also: printBase, <<

7.7.3 <<

Usage:

```
#include <PulGARP.h>
ostream& operator << (ostream& ostr, PulGARP& PG)
```

Description:

The operator << adds the GARP parameters specified as an argument ***PG*** to the output stream ***ostr***.

Return Value:

None.

Example:

```
Garp GP;
GP.read("filein.pset");
cout << GP;
```

See Also: print, printBase

7.8 Auxiliary Functions

7.8.1 channel

Usage:

```
#include <PulGARP.h>
String PulGARP::channel( )
```

Description:

The function ***channel*** returns a string indicating the isotope channel the pulse waveform is applied on.

Return Value:

The function returns a string.

Example:

```
#include <PulGARP.h>
PulGARP PW = GARP(600.0, “13C”);                    // GARP-1 Waveform
cout << “\n\tGARP Decouple On “ << PW.channel();// Output channel
```

See Also:

7.8.2 steps

Usage:

```
#include <PulGARP.h>
int PulGARP::steps( )
```

Description:

The function ***steps*** returns the number of individual steps defined in the pulse waveform.

Return Value:

The function returns an integer.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.3 cycles

Usage:

```
#include <PulGARP.h>
int PulGARP::cycles( )
```

Description:

The function ***cycles*** returns the number of individual cycle steps defined in the pulse waveform.

Return Value:

The function returns an integer.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.4 name

Usage:

```
#include <PulGARP.h>
String PulGARP::name( )
```

Description:

The function ***name*** returns the name of the pulse waveform.

Return Value:

The function returns a string.

Example:

7.8.5 values

Usage:

```
#include <PulGARP.h>
row_vector PulGARP::values( )
```

Description:

The function ***values*** returns a row_vector containing values which define the pulse waveform steps. The ith vector value contains the values {γB1, ϕ}, where the real component is the rf-field strength in Hz, and the imaginary component is the rf-phase in degrees (or radians).

Return Value:

The function returns a row vector.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.6 value

Usage:

```
#include <PulGARP.h>
complex PulGARP::value(int i)
```

Description:

The function ***value*** returns a compex number for the values which define the pulse waveform step ***i***. The value contains the number $\{\gamma B1, \phi\}$, where the real component is the rf-field strength in Hz, and the imaginary component is the rf-phase in degrees (or radians).

Return Value:

The function returns a complex number.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.7 phase

Usage:

```
#include <PulGARP.h>
double PulGARP::phase(int i)
```

Description:

The function ***phase*** returns the value of the rf-field phase at pulse waveform step ***i*** in degrees (or radians).

Return Value:

The function returns a double.

7.8.8 strength

Usage:

```
#include <PulGARP.h>
double PulGARP::strength(int i)
```

Description:

The function ***strength*** returns the value of the rf-field amplitude at pulse waveform step ***i*** in Hz.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.9 length

Usage:

```
#include <PulGARP.h>
double PulGARP::length(int i)
```

Description:

The function ***length*** returns the length of th pulse waveform in seconds.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.10 steplength

Usage:

```
#include <PulGARP.h>
double PulGARP::steplength(int i)
```

Description:

The function ***steplength*** returns the length of an individual pulse waveform step in seconds.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
```

7.8.11 cyclelength

Usage:

```
#include <PulGARP.h>
double PulGARP::cyclelength(int i)
```

Description:

The function ***cyclelength*** returns the length of the pulse waveform cycle in seconds.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.12 scyclelength

Usage:

```
#include <PulGARP.h>
double PulGARP::scyclelength(int i)
```

Description:

The function ***scyclelength*** returns the length of th pulse waveform supercycle in seconds.

Return Value:

The function returns a double.

Example:

```
#include <PulGARP.h>
```

See Also:

7.8.13 FZ

Usage:

```
#include <PulGARP.h>
int PulGARP::FZ( )
```

Description:

The function ***FZ*** returns the z-axis spin operator associated with the pulse waveform. This operator will be selective for the isotope which the pulse waveform affects.

Return Value:

The function returns an operator.

Example:

```
#include <PulGARP.h>
```

See Also:

7.9 Description

7.9.1 Introduction

The functions in module ***PulGARP*** and Class ***GARP*** (contained in module ***PulGARP***), is designed to facilitate the use of GARP[1] pulse trains in GAMMA NMR simulation programs. In GAMMA, as in an NMR experiment, we should like to use GARP pulse trains as individual steps in a general pulse sequence, including use in variable delays as part of multi-dimensional experiments and/or use in pulse trains during acquisition steps.

7.9.2 GARP Parameters

A variable of type GARP contains only primitive parameters: e consider a pulse waveform as involving four basic features: 1.) A ***# steps***, 2.) An ***rf-field strength***, 3.) An ***rf-phase*** 4.) An ***rf-offset***. These value can be used to completely determine how to set up composite pulses such as that used in a GARP-1 pulse train.

7.9.3 Basic GARP Waveform

GARP sequences are based on a 25 step composite pulse. The pulses are applied with the same rf-strength but vary in their applied length and oscillate phase between ϕ and $\phi+\pi$. The details are shown in the following figure.

Basic GARP 25 Step Sequence

Step	Angle	Step	Angle	Step	Angle
1	30.5	9	134.5	17	258.4
2	55.2	10	256.1	18	64.9
3	257.8	11	66.4	19	70.9
4	268.3	12	45.9	20	77.2
5	69.3	13	25.5	21	98.2
6	62.2	14	72.7	22	133.6
7	85.0	15	119.5	23	255.9
8	91.8	16	138.2	24	65.6
				25	53.4

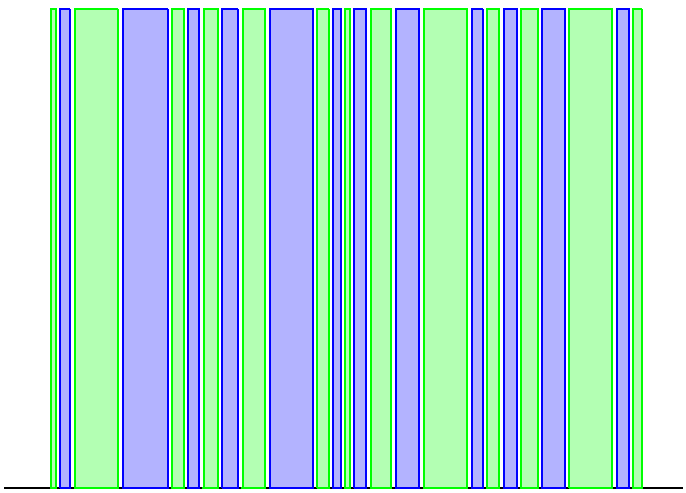

Figure 7-1 The basic 25-step GARP waveform. The blue steps indicate pulse that are applied with a 180 degree phase shift, as indicate by a bar in the table listing. The program which produced this plot can be found at the end of this chapter, GarpWF0.cc on page 130.

1. For information on GARP see the article by Shaka, Barker and Freeman in *J. Magn. Reson.*, **64**, 547-552 (85). GARP = Globally optimized Alternating-phase Rectangular Pulses.

7.9.4 GARP-1 Pulse Cycle

The GARP-1 decoupling sequence repeatedly uses the GARP 25 step composite pulse but changes the overall phase in a WALTZ-4 . Thus for GARP-1 we have

GARP-1 Decoupling Sequence

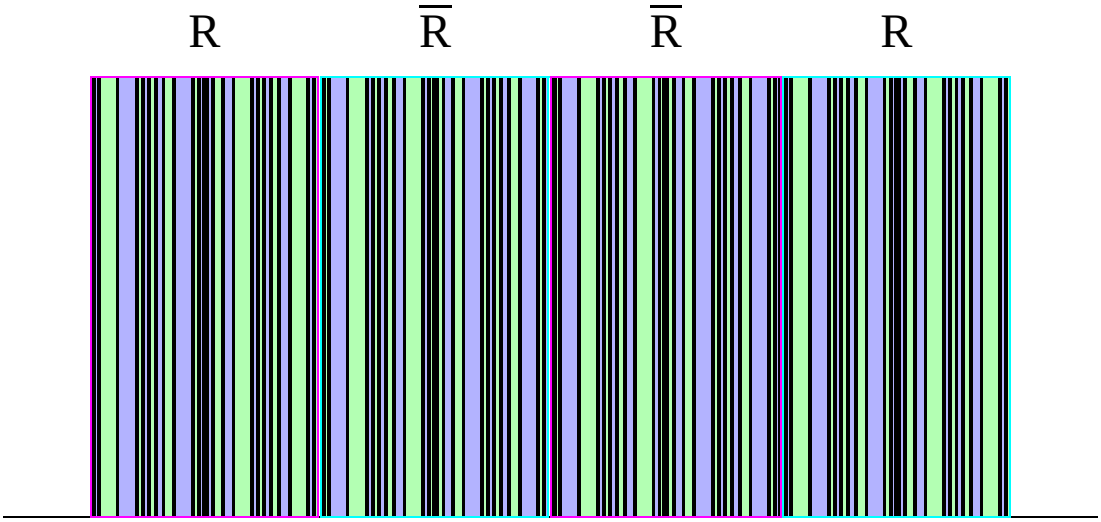

Figure 7-1 The GARP-1 decoupling sequence. Each 25-step GARP waveform is repeated with the 4-step phase adjustment overlaid: 0, 180, 180, 0. These are designated R, $\overline{R}$, $\overline{R}$, R respectively. The 4 waveforms (100 steps) of GARP-1 are repeated as long as the decoupling sequence is applied.

The GARP 25-step waveforms (composite pulses) are continuously applied with the same rf-strength but will change phase between ϕ and $\phi+\pi$. As is typical in such sequence, the cycle phases are changed in a 4-step sequence: 0, 180, 180, 0. Thus the first and last 25 steps of the GARP-1 cycle will be identical as will the second and third 25 steps. But these two types are 180 degrees out of phase.

7.10 GARP Parameters

This section describes how an ASCII file may be constructed that is self readable by a GARP variable. The file can be created with an editor of the users choosing and is read with the GARP member function “read”. This provides for an extremely flexible and program independent means of implementing GARP in NMR simulations.

The GARP (ASCII) input file is scanned for the specific parameters which specify the pulse-delay parameters[1]: delay length, rf-length, rf-strength, rf-phase, rf-offset, pulse angle, and the number of GARP step. These parameters are recognized by certain keywords, as shown in the following table.

Table 2: Spin System Parameters

Parameter Keyword	Assumed Units	Examples Parameter (Type) : Value - Statement
GARPgamB1	Hz	GARPgamB1 (1) : 600.0 - Field Strength (Hz)
GARPiso	none	GARPiso (2) : 19F - GARP rf pulse channel
GARPphi	degrees	GARPphi (1) : 2.0 - GARP rf phase (deg)
GARPstps	none	GARPstps (0) : 20 - GARP pulse-delay steps

The order in which these parameters reside in the ASCII file is of no consequence.

The format of each parameter is quite simple and general for all GAMMA parameters. At the beginning of a line the parameter keyword is written followed by an optional index number in parenthesis. This is then followed by one or more blanks and then an integer in parentheses. The integer corresponds to the type of parameter value: 0 = integer, 1 = floating point, or 2 = string. Following the parenthesis should be at least one blank then a colon to indicate the parameter value follows. The parameter value is then written followed by some blanks then a hyphen followed by an optional comment.

There is one major restriction; keywords and string parameters cannot contain blanks. For example, v (0) is unknown, v(0) is. The string value 19 F is unknown, 19F is fine. If multiple GARP pulse-delay steps need to be defined in the same file then simply put an index on all parameters associated with a desired GARP and read the parameters using that index.

To read the file, see the documentation for function read (or ask-read). There is also an example program readsystem.cc provide at the end of this Chapter which should indicate how the file is read. Each of the possible spin system input parameters is now described in more detail.

1. Note that the ASCII file must contain viable parameters in GAMMA format. Indeed, the file is a GAMMA parameter set and, as such, may contain any amount of additional information along with the valid GARP parameters.

Channel: GARPiso

This parameter is optional. It will define which isotope channel the GARP rf-pulse will be applied on. If no channel is specified GAMMA will assume that all spins in the system being treated are affected by the rf. Thus if no channel is specified and GARP is utilized in an NMR simulation the system should be homo-nuclear or the same GARP should be desired on all channels (same offset, phase, etc.)

Channel: GARPphi

This parameter is optional. It will define the rf-phase of the GARP pulse. If no phase is specified it will be taken to be zero.

Pulse Length: GARPtp

The parameters { GARPang, GARPgamB1, GARPtp } work together. GARPtp will set the pulse length if either GARPang and/or GARPgamB1 have also been specified. If only GARPtp has been specified amongst the three an error will result when reading these parameters to define GARP.

Pulse Strength: GARPgamB1

The parameters { GARPang, GARPgamB1, GARPtp } work together. GARPgamB1 se the pulse strength if either GARPang or GARPgamB1 have also been specified. If only GARPgamB1 is specified amongst the three an error will result when reading these parameters to define GARP. If all three parameters have been specified then GARPgamB1 will be ignored, the strength set by { GARPang, GARPtp }

Sync Frequency: GARPF

This parameter sets the GARP frequency, i.e. the pulse-delay repetition rate. Thus, users may specify the specific frequency that GARP will affect the strongest. The parameter will override any delay time set by the parameter GARPtp. The combined length of the GARP pulse and delay will be set to 1/GARPF.

Delay Length: GARPtp

This parameter sets the GARP delay length, independent of the GARP pulse length. If GARPF exists then this parameter will not be used.

7.11 GARP Examples

7.11.1 Reading GARP Parameters

To keep GAMMA programs using GARP sequences versatile, users will want to keep all GARP specifications undetermined in the code. As the program runs, GARP settings are either specified interactively and/or read in from an external ASCII (parameter) file. This section gives examples of the latter case. The figure below shows an ASCII parameter file on the left and some GAMMA program code on the right.

Reading GARP Parameters

```
GARPphi          (1) : 90.0     - GARP overall rf-phase (deg)
GARPiso          (2) : 1H       - GARP pulse channel
GARPgamB1        (1) : 983.0    - GARP pulse strength (Hz)
GARPiso(3)       (2) : 19F      - GARP pulse channel
GARPgamB1(3)     (2) : 2045.9   - GARP pulse strength (Hz)
```

```
GARP GP;                              // Declare GARP parameters
GP.read("GARP.pset");                 // Read GARP from file
GP.read("GARP.pset", 3);              // Read GARP from file
GP.ask_read(argc, argv, 1);           // Read GARP from file
GP.ask_read(argc, argv, 2, 3);        // Read GARP from file
```

Figure 7-2 The basic 25-step GARP waveform. The blue steps indicate pulse that are applied with a 180 degree phase shift, as indicate by a bar in the table listing. The program which produced this plot can be found at the end of this chapter, GarpWF0.cc on page 130.

The ASCII parameter file (on the left) is taken to be called "GARP.pset" and is read in by the program code. Thus one can change the GARP parameters independent of the GAMMA code. The ASCII file format is typical of GAMMA parameter sets: The line ordering is of no consequence, the column spacing is not important, the end "- comments" can be left off, and additional lines of text or parameters may be included.

The GAMMA code is color coded with the parameters they read in the previous figure. Thus the second line (blue) will read the blue parameters and set up GARP with a strength of 983 Hz on the proton channel with an overall phase of 90 degrees. Similarly, the next line (green) will read the parameters colored green from the same ASCII file but sets up GARP with a strength of 2.0459 kHz on the 19F channel (no phase, no offset).

The next line will interactively ask the user to supply a filename where the program can get to GARP parameters. This filename (in this case "GARP.pset") will be prompted for unless the user specifies the file on the command line when the program is executed. The following line does the same but reads the GARP parameters indexed with a "3" from the file.

Using a combination of these commands, the user has complete flexibility in defining one or more GARP sequences in the same GAMMA program. The GARP parameters can be easily changed by either changing their values in the ASCII file and/or changing the filename given to the program. See the section of GARP parameters to see which parameters can be used in setting up GARP sequences. See the other programs in this chapter for full examples GAMMA programs using them.

7.11.2 GARP-1 Decoupling

In this section we shall produce a simple 1D NMR spectrum under GARP-1 decoupling. A hard 90 pulse will be applied to a chosen spin system on the acquisition channel. Then GARP-1 will be applied on the decoupler channel during acquisition. The resulting FID will be apodized and Fourier transformed, the NMR spectrum put on screen using Gnuplot. Note that relaxation and exchange effects will be ignored in this simulation. The code for simple GARP-1 decoupling is shown below:

```
/* GARPdec0.cc ************************************************************************
**                                                                                      **
**                        GAMMA Decoupling Example Program                              **
**                                                                                      **
**  This program uses the class GARP to perform a simple decoupling                     **
**  simulation.  A hard ideal pulse will be applied to an input spin                    **
**  system.  Subsequently, an acquisition will be performed with GARP-1                 **
**  decoupling applied on a specified channel.                                          **
**                                                                                      **
** Author:      S.A. Smith                                                             **
** Date:        3/9/98                                                                 **
** Update:      3/9/98                                                                 **
** Version:     3.5.4                                                                  **
** Copyright:   S. Smith.  Modify this program as you see fit for personal             **
**              use, but you must leave the program intact if redistrubuted            **
**                                                                                      **
*****************************************************************************************
**/

#include <gamma.h>
#
main(int argc, char* argv[])
 {
 cout << "\n\t\t\t\tGARP Decoupling\n\n";
 int qn = 1;                                          // Query index
 spin_system sys;                                     // Declare a spin system
 GARP GP;                                             // GARP parameters
 String filename = sys.ask_read(argc,argv,qn++);      // Ask for/read in the system
 cout << sys;                                         // Have a look (for setting SW)
 GP.read(filename);                                   // Read in GARP parameters
 PulCycle PCyc = GP.CycGARP1(sys);                    // Construct GARP-1 cycle
 String IsoD = sys.symbol(0);                         // Detection/pulse channel
 if(sys.heteronuclear())                              // If heteronuclear system
   query_parameter(argc, argv, qn++,                  // ask for detection channel
       "\n\tDetection Isotope? ", IsoD);
 double SW;                                           // Spectral width
 query_parameter(argc, argv, qn++,                    // Get desired spectral width
       "\n\tSpectral Width (Hz)? ", SW);
 int npts = 1024;                                     // Block size (must be base 2)
 query_parameter(argc, argv, qn++,                    // Get block size
       "\n\tBlock Size? ", npts);
 double lwhh = 1.0;                                   // Half-height linewidth
 query_parameter(argc, argv, qn++,                    // Ask for apodization strength
       "\n\tApodization (Hz)? ", lwhh);
 double td = 1/SW;                                    // Set dwell time
 double R = (lwhh/2)*HZ2RAD;                          // Set apodization rate
 double tt = (npts-1)*td;                             // Total FID length
 gen_op H = Ho(sys);                                  // Set isotropic Hamiltonian
 gen_op Det = Fm(sys, IsoD);                          // Set detection operator to F-
 gen_op sigma0 = sigma_eq(sys);                       // Set density mx equilibrium
 gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.);        // This is 1st 90 pulse
 row_vector data = PCyc.FID(npts,td,Det,sigmap);      // Perform acquisition under GARP-1
 row_vector exp = Exponential(npts,tt,0.0,R,0);       // Here is an exponential
 row_vector fidap = product(data,exp);                // Apodized FID
 data = FFT(fidap);                                   // Transformed FID -> spectrum
 GP_1D("spec.asc", data, 0, -SW/2, SW/2);             // Output ASCII file
 GP_1Dplot("spec.gnu", "spec.asc");                   // Plot to screen using Gnuplot
 }
```

The first half of this program simply sets the parameters up interactively. Note that both the spin system and the GARP parameters are contained in the same file who's name the user must specify.

The second half of the program does the simulation. The FID is acquired using the pulse cycle function "FID" and the pulse cycle has been set to GARP-1. The last few lines apodize and transform the FID then spit the plot out on the screen.

Addition of relaxation & exchange effects and/or changing the 1st pulse to non-ideal will require only minor modifications of a few lines.

The input parameter (ASCII) file is listed on the following page. It contains parameters for both the spin system and the GARP settings.

Example of a GARP decoupling input file (GARPdec.sys)

```
SysName      (2) : GARP     - Name of the Spin System
NSpins       (0) : 4        - Number of Spins in the System
Iso(0)       (2) : 1H       - Spin Isotope Type
Iso(1)       (2) : 1H       - Spin Isotope Type
Iso(2)       (2) : 1H       - Spin Isotope Type
Iso(3)       (2) : 13C      - Spin Isotope Type
v(0)         (1) : 105.0    - Chemical Shifts in Hz
v(1)         (1) : -174.32  - Chemical Shifts in Hz
v(2)         (1) : 15.0     - Chemical Shifts in Hz
v(3)         (1) : 0.0      - Chemical Shifts in Hz
J(0,1)       (1) : 10.0     - Coupling Constants in Hz
J(0,2)       (1) : 7.9      - Coupling Constants in Hz
J(0,3)       (1) : 22.0     - Coupling Constants in Hz
J(1,2)       (1) : 2.8      - Coupling Constants in Hz
J(1,3)       (1) : 32.0      - Coupling Constants in Hz
J(2,3)       (1) : 18.3     - Coupling Constants in Hz
Omega        (1) : 400      - Spect. Freq. in MHz (1H based)

GARPphi      (1) : 0        - GARP pulse phase (deg)
GARPiso      (2) : 13C      - GARP pulse channel
GARPgamB1    (1) : 1500.0   - GARP pulse strength (Hz)
```

When the program (GARPdec0.cc) is compiled its execution will produce a plot on screen if the Gnuplot program is available. Assuming the executable is called a.out, the following command will produces a spectrum:

a.out GARPdec.sys 1H 500 1024 .5

The command “a.out” alone will prompt you for input values. Had you input the above command (or parameters) the spectrum should appear as shown in the following figure.

^{13}C Decoupled Spectrum Using GARP-1

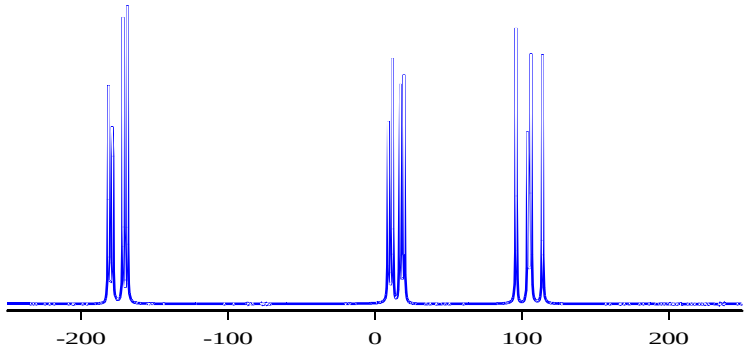

Figure 7-3 The spectrum produced using the program GARPdec0.cc with input parameter file GARPdec.sys. The decoupler was applied to the 13C channel with a 1.5 kHz field strength. Detection was on the proton channel. 1K data points were collected using a spectral width of 500 Hz. The data was processed with a 0.5 Hz line-broadening.

By either editing the input file or specifying a different input file, the spectrum can be radically altered. For example, by setting the GARP pulse strength to zero one obtains the following spectrum.

^{13}C Coupled Spectrum, Zero Strength GARP-1

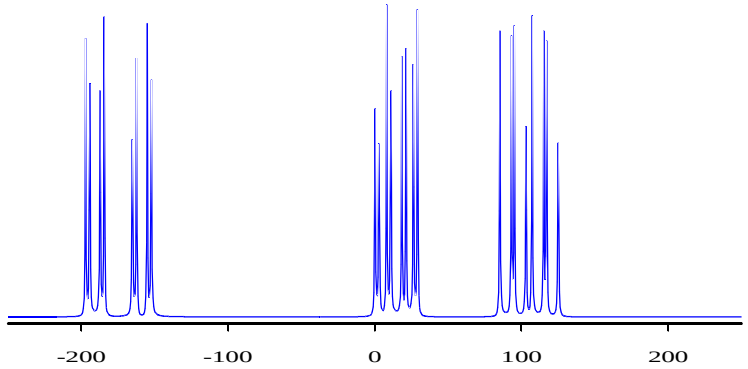

Figure 7-4 Same as previous figure but with no decoupler field strength.

7.11.3 GARP-1 Decoupling vs. Field

We can readily modify the previous program to loop over differing rf-field strengths and determine how well GARP-1 does as decoupling. In this case we will just read in a series of gB1 values from an external ASCII file and loop over them producing a 1D spectrum at each value. We'll spit out all the spectra in a single stack plot.

```
/* GARPdec1.cc ***************************************************************************
**                                                                                      **
**                          GAMMA Decoupling Test Program                               **
**                                                                                      **
**  This program uses the class GARP to perform a simple decoupling                     **
**  simulation.  A hard ideal pulse will be applied to an input spin                    **
**  system.  Subsequently, an acquisition will be performed with GARP-1                 **
**  decoupling applied on a specified channel.                                          **
**                                                                                      **
** Author:      S.A. Smith                                                             **
** Date:        3/11/98                                                                **
** Update:      3/11/98                                                                **
** Version:     3.5.4                                                                  **
** Copyright:   S. Smith.  You can modify this program as you see fit                  **
**              for personal use, but you must leave the program intact                 **
**              if you re-distribute it.                                                **
**                                                                                      **
******************************************************************************************/

#include <gamma.h>

main(int argc, char* argv[])
 {
 cout << "\n\t\t\t\tGARP Decoupling Vs. Decoupler Strength\n\n";
 int qn = 1;                                        // Query index
 spin_system sys;                                   // Declare a spin system
 GARP GP;                                           // GARP parameters
 String filename;                                   // Input filename
 filename = sys.ask_read(argc,argv,qn++);           // Ask for/read in the system
 cout << sys;                                       // Have a look (for setting SW)
 GP.read(filename);                                 // Read in GARP parameters
 cout << GP;
 PulCycle PCyc;

 query_parameter(argc, argv, qn++,                  // Ask for field strength file
   "\n\tFile of Field Strengths? ", filename);
 int N;                                             // Number of field strengths
 double* gB1s = GetDoubles(filename, N);            // Get array of field strengths
 String IsoD = sys.symbol(0);                       // Detection/pulse channel
 if(sys.heteronuclear())                            // If heteronuclear system
  query_parameter(argc, argv, qn++,                 // ask for detection channel
        "\n\tDetection Isotope? ", IsoD);
 double SW;                                         // Spectral width
 query_parameter(argc, argv, qn++,                  // Get desired spectral width
        "\n\tSpectral Width (Hz)? ", SW);
 int npts = 1024;                                   // Block size (must be base 2)
 query_parameter(argc, argv, qn++,                  // Get block size
           "\n\tBlock Size? ", npts);
 double lwhh = 3.0;                                 // Half-height linewidth
 query_parameter(argc, argv, qn++,                  // Ask for apodization strength
          "\n\tApodization (Hz)? ", lwhh);
 double td = 1/SW;                                  // Set dwell time
 double R = (lwhh/2)*HZ2RAD;                        // Set apodization rate
 double tt = (npts-1)*td;                           // Total FID length
 gen_op H = Ho(sys);                                // Set isotropic Hamiltonian
 gen_op Det = Fm(sys, IsoD);                        // Set detection operator to F-
 gen_op sigma0 = sigma_eq(sys);                     // Set density mx equilibrium
 gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.);      // This is 1st 90 pulse
 row_vector data, exp, fidap;
 matrix datamx(N,npts);
 for(int i=0; i<N; i++)
  {
  GP.strength(gB1s[i]);                             // Set GARP field strength
  PCyc = GP.CycGARP1(sys);                          // Set GARP-1 pulse cycle
  data = PCyc.FID(npts,td,Det,sigmap);              // Perform acquisition
  exp = Exponential(npts,tt,0.0,R,0);               // Here is an exponential
  fidap = product(data,exp);                        // Apodized FID
  data = FFT(fidap);                                // Transformed FID -> spectrum
  datamx.put_block(i,0,data);
  }
 double Nm1 = double(N-1);
 String AF("stk.asc");
 String GF("stk.gnu");
 GP_stack(AF, datamx, 0,1,N,0.0,Nm1);
 GP_stackplot(GF, AF);
 FM_stack("stk.mif", datamx, 1.5, 1.5, 1);
 }
```

The modifications from the previous program are obvious. An external ASCII file is used to specify a list of decoupler field strengths and these are read into the program. These fields are looped over, a new spectrum computed at each decoupler strength. The spectra are put into a matrix which is given to the Gnuplot routines for display as a stack plot on screen. In addition, the stack plot

is output in FrameMaker MIF format for incorporation into documents in an editable form. The latter is shown in the following figure.

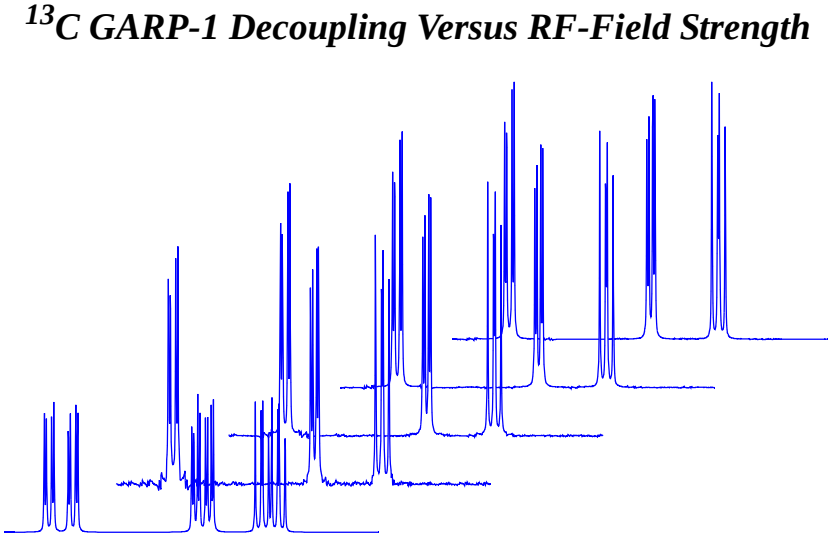

Figure 7-5 Proton spectra produced using the program GARPdec1.cc with input parameter file GARPdec.sys and decoupler strength file GARPdecBs. The decoupler was applied to the 13C channel with field strengths shown. Detection was on the proton channel. 1K data points were collected using a spectral width of 500 Hz. The data was processed with a 1.0 Hz line-broadening.

When the program (GARPdec1.cc) is compiled its execution will produce a stack plot on screen if the Gnuplot program is available. Assuming the executable is called a.out, the following command will produce the plot shown in the previous figure:

a.out GARPdec.sys GARPdecBs 1H 500 1024 1.0

The ASCII file GARPdecBs contains a list of rf-field strengths (in Hz) that the program used. The file has a single field strength per line and is shown next.

GARP decoupling rf-field input file (GARPdecBs)

```
0
200
400
600
800
```

Unlike GAMMA parameter set files (such as GARPdec.sys) this file is simple ASCII and cannot have anything other than a single floating point or integer value per line. No additional comments may be included.

7.11.4 GARP-1 Decoupling Profile

In this section we shall attempt to produce a GARP-1 decoupling profile. A hard 90 pulse will be applied to a simple heteronuclear spin system on the acquisition channel. Then GARP-1 will be applied on the decoupler channel during acquisition. The resulting FID will be apodized and Fourier transformed. This pulse-delay process will be repeated for differing offsets on the decoupler channel. Each spectrum will be plotted with its center at the offset frequency to produce the profile.

The really no significant differences between this and our previous calculations. To determine a profile one uses the simplest spin system (here a two spin heteronuclear system). The 1D spectrum is recalculated after either moving the decoupler rf offset or, equivalently, moving all decoupler isotope channel chemical shifts. The spectra are all just put into a single vector, offset so their respective centers are set to be referenced to the decoupler offset value.

For fun, we'll design the GAMMA program to allow for the reproduction of Fig 1c. (bottom) in the original GARP-1 publication by Shaka, Barker, and Freeman (page 550). In fact, here are the GAMMA simulation results.

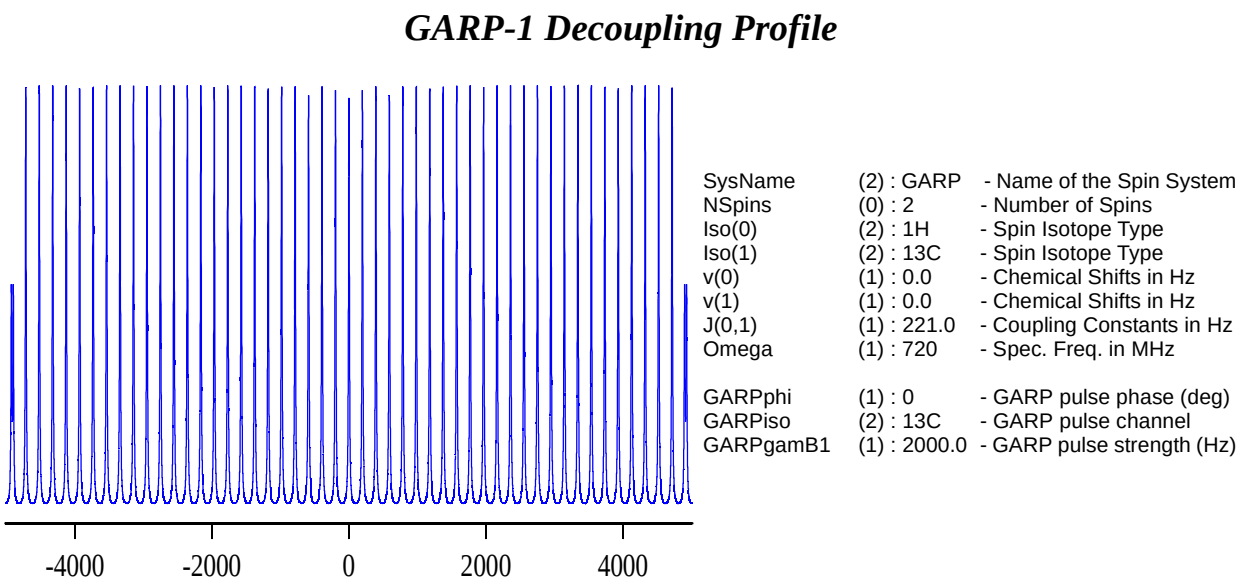

Figure 7-6 GARP-1 decoupling profile. Decoupling was performed on the carbon channel in a ^{13}C-^{1}H two spin system. The decoupling rf-field strength was set to 2 kHz and the scalar coupling to 221 Hz. A linebroadening of 1.5 Hz was used in processing the spectra. The block size was 1K and the offset increment set to 200 Hz. These parameters were use to mimic the Shaka et. al paper. The text at the right is the file which was fed into the simulation program.

The agreement is excellent. The code for a "synchronous" GARP-1 decoupling profile is shown on the next page. This program sets a spectral width such that acquisition points are taken only after an even number of GARP-1 cycles (or at least an even number of GARP 25-step waveforms). In examining the program note that there are very few lines that have much to do with GARP. A quick replacement of a couple of lines would make this use MLEV or WALTZ or ..... Additionally we could adjust it let the user select among decoupling sequences. Even better, we can perform a slight adjustment and include the effects of relaxation and/or exchange.

```
/* GARPprof1.cc ***************************************************************************
**                                                                                       **
**                 GAMMA Decoupling Test Program                                         **
**                                                                                       **
** This program uses the class GARP to perform a simple decoupling                      **
** simulation.  A hard ideal pulse will be applied to a simple two spin  hetero-         **
** nuclear system.  Subsequently, an acquisition will be performed with                  **
** GARP-1 decoupling applied on one the channel which is not being                       **
**  idetected.  This process will be repeated over a range of decoupler offsets.        **
** The result is a GARP-1 decoupler profile and will be plotted on screen if             **
** Gnuplot is available on the system.  The profile is also output inMIF.                **
**                                                                                       **
** Author:      S.A. Smith                                                               **
** Date:        3/9/98                                                                   **
** Update:      3/9/98                                                                   **
** Version:     3.5.4                                                                    **
** Copyright:  S. Smith.  You can modify this program as you see fit for your            **
**                  use, but you must leave the program intact if distrubuted.           **
**                                                                                       **
*******************************************************************************************/

#include <gamma.h>

main(int argc, char* argv[])
 {
 cout << "\n\t\t\t\tGARP Decoupling Profile\n\n";
//              Read In Spin System & GARP Parameters
int qn = 1;                                     // Query index
 spin_system sys;                               // Declare a spin system
 GARP GP;                                       // GARP parameters
 String filename;                               // Input filename
 filename = sys.ask_read(argc,argv,qn++);       // Ask for/read in the system
 cout << sys;                                   // Have a look (for setting SW)
 if(sys.spins()!=2 || sys.homonuclear())
   cout << "\n\tWarning! This program has been"
        << " set up for a two spin heteronuclear"
        << " system.\n Results on other systems"
        << " can be unpredictable.....";
 GP.read(filename);                             // Read in GARP parameters
//              Set Acquistion and Profile Parameters
String IsoD = sys.symbol(0);                    // Detection/pulse channel
 String IsoG = GP.channel();                    // Decoupler channel
 if(IsoD == IsoG)                               // Try and set channel to
   IsoD = sys.symbol(1);                        // not be the decoupling one
 double SW;                                     // Spectral width
 query_parameter(argc, argv, qn++,              // Get desired spectral width
          "\n\tSpectral Width (Hz)? ", SW);
 int npts = 1024;                               // Block size (must be base 2)
 query_parameter(argc, argv, qn++,              // Get block size
          "\n\tBlock Size? ", npts);
 double lwhh = 3.0;                             // Half-height linewidth
 query_parameter(argc, argv, qn++,              // Ask for apodization strength
          "\n\tApodization (Hz)? ", lwhh);
 int NO = 30;                                   // # Of Offsets (on each side)
 query_parameter(argc, argv, qn++,              // Get # offsets
  "\n\tNumber of Positive Decoupler Offsets? ", NO);
 double offset;
 query_parameter(argc, argv, qn++,              // Get # offsets
  "\n\tDecoupler Offset Per Step (Hz)? ", offset);
//              Set Up Variables Consistent Through All Offsets
double R = (lwhh/2)*HZ2RAD;                     // Set apodization rate
 gen_op Det = Fm(sys, IsoD);                    // Set detection operator to F-
 gen_op sigma0 = sigma_eq(sys);                 // Set density mx equilibrium
 gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.);  // This is 90 detection pulse
 row_vector data(npts);                         // Block for acquisiton
//              Set Up Variables Global Over Full Profile
row_vector profile((2*NO+1)*npts, complex0);    // Block for profile
 double totaloff = double(NO)*offset;           // Total offset at end
 row_vector fidap;                              // Block for apodized FID
 PulCycle PCyc = GP.CycGARP1(sys);              // Empty GARP-1 pulse cycle
 SW = PCyc.FIDsync(SW);                         // Synchronize dwell times
 double td = 1/SW;                              // Set dwell time
 double tt = (npts-1)*td;                       // Total FID length
 row_vector exp=XExponential(npts,tt,0.0,R,0);  // Block for apodization
 PCyc.print(cout, 1);
//              Loop Over Offsets, Calculate Profile
int K =0;                                       // Point index in profile
 sys.offsetShifts(-NO*offset, IsoG);            // Set 1st profile offset
 for(int ov=-NO; ov<=NO; ov++)                  // Loop over offsets
  {
  PCyc  = GP.CycGARP1(sys);                     // GARP-1 cycle this offset
  data  = PCyc.FID(npts,td,Det,sigmap);         // Acquisition this offset
  fidap = product(data,exp);                    // Apodized FID this offset
  data  = FFT(fidap);                           // Spectrum this offset
  profile.put_block(0, K, data);                // Put spectrum in profile
  sys.offsetShifts(offset, IsoG);               // Move system to next offset
  K += npts;                                    // Adjust profile point index
  }

 double F = totaloff + SW/2;                    // Final plot frequency
 GP_1D("prof.asc", profile, 0, -F, F);          // Output profile ASCII data
 GP_1Dplot("prof.gnu", "prof.asc");             // Plot to screen using Gnuplot
 FM_1D("prof.mif", data,14,14,-F, F);           // Plot in FrameMaker MIF
 }
```

7.11.5 GARP-1 Modified Decoupling Profile

Suppose that we wish to follow along with the original GARP publication and reproduce all of the decoupling profiles simulated therein. How would we go about doing so? GAMMA builds up pulse cycles from a combination of a pulse waveform and a cycle overlay. For GARP-1 there is a function supplied in this module (PulGarp) which simply returns a pulse cycle based on the 25-step GARP waveform and the 4-step $R\overline{R}RR$ cycle. For decoupling sequences which are not available through an existing GAMMA function the user can simply build his/her own.

Let's consider the "90%" GARP-1 simulation in the paper (Fig. 1c top). This was performed to examine how well GARP-1 performs when the pulses are inadvertently mis-calibrated. The authors intentionally set the decoupler field strength to be 90% of what is required for the proper pulse angles in the GARP sequence. To accomplish this in GAMMA we will modify the previous simulation by explicitly building our own (bad) GARP-1 pulse cycle. Rather that use the provided GARP-1 function, we will just scale the provided GARP waveform rf-field strength by 90% then build a modified GARP-1 pulse cycle with the scaled waveform. Everything else will be very much the same, i.e the same input values and the same input file. For convenience we will add an interactive request for the scaling factor (which will be set to 0.90 to reproduce the paper simulation).

First the simulation output.

GARP-1 90% RF-Power Decoupling Profile

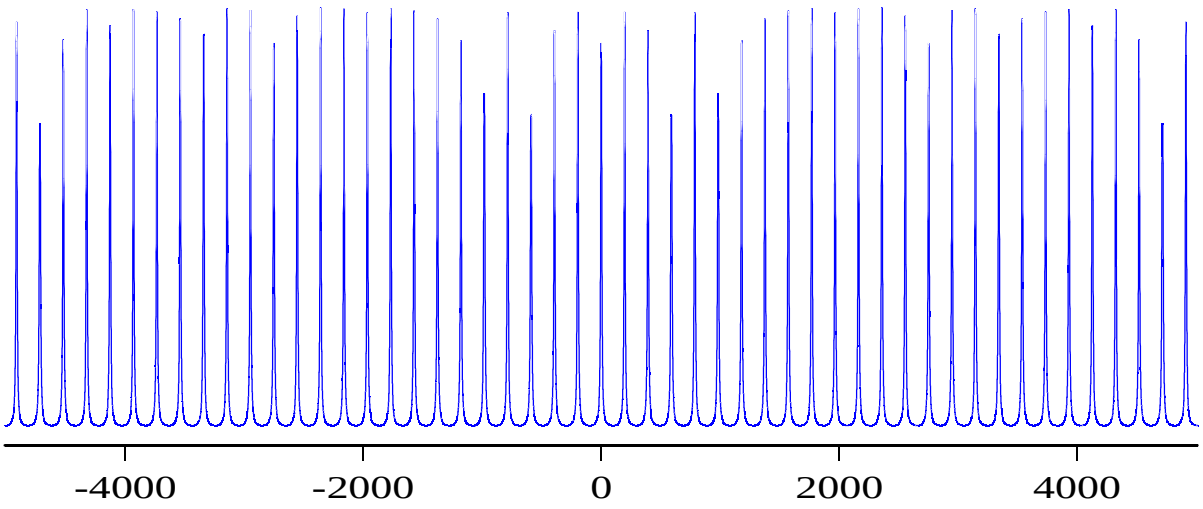

Figure 7-7 GARP-1 decoupling profile @ 90% RF power. Decoupling was performed on the carbon channel in a ^{13}C-^{1}H two spin system. The decoupling rf-field strength was set to 2 kHz x 0.90 after calibrating the pulse lengths for 2 kHz. The scalar coupling to 221 Hz. A linebroadening of 1.5 Hz was used in processing the spectra. The block size was 1K and the offset increment set to 200 Hz.

Again this is in excellent agreement with the GARP-1 paper. The simulation program is shown on the following page. I have placed the important code changes in blue so that it is evident how users can construct their own pulse cycles. I don't know what the other decoupling sequences are in the paper cited, so I won't bother looking at them. You can if you like though. I ran the program with the command "a.out GARPprof.sys 25 1024 1.5 25 200 0.9" where a.out is the executable.

```
/* GARPprof2.cc ***************************************************************************
**                                                                                       **
**                          GAMMA Decoupling Test Program                                **
**                                                                                       **
**  This program uses the class GARP to perform a decoupling profile                     **
**  simulation.  A hard ideal pulse will be applied to a simple two                      **
**  spin  heteronuclear system.  Subsequently, an acquisition will be                    **
**  performed with GARP-1 decoupling applied on one the channel which                    **
**  is not being detected.  This process will be repeated over a range                   **
**  of decoupler offsets.  The result is a GARP-1 decoupler profile.                     **
**                                                                                       **
** Author:      S.A. Smith                                                             **
** Date:        3/9/98                                                                 **
** Update:      3/9/98                                                                 **
** Version:     3.5.4                                                                  **
** Copyright:  S. Smith.  You can modify this program as you see fit for your          **
**                  use, but you must leave the program intact if distrubuted.           **
**                                                                                       **
*******************************************************************************************/

#include <gamma.h>

main(int argc, char* argv[])
 {
 cout << "\n\t\t\t\tGARP Decoupling Profile\n\n";
//              Read In Spin System & GARP Parameters
int qn = 1;                                     // Query index
 spin_system sys;                               // Declare a spin system
 GARP GP;                                       // GARP parameters
 String filename;                               // Input filename
 filename = sys.ask_read(argc,argv,qn++);       // Ask for/read in the system
 cout << sys;                                   // Have a look (for setting SW)
 if(sys.spins()!=2 || sys.homonuclear())
   cout << "\n\tWarning! This program has been set up for a two spin heteronuclear "
        << " system.\n Results on other systems can be unpredictable.....";
 GP.read(filename);                             // Read in GARP parameters
//              Set Acquistion and Profile Parameters
 String IsoD = sys.symbol(0);                   // Detection/pulse channel
 String IsoG = GP.channel();                    // Decoupler channel
 double SW;                                     // Spectral width
 query_parameter(argc, argv, qn++,              // Get desired spectral width
                 "\n\tSpectral Width (Hz)? ", SW);
 int npts = 1024;                               // Block size (must be base 2)
 query_parameter(argc, argv, qn++,              // Get block size
                         "\n\tBlock Size? ", npts);
 double lwhh = 3.0;                             // Half-height linewidth
 query_parameter(argc, argv, qn++,              // Ask for apodization strength
          "\n\tApodization (Hz)? ", lwhh);
 int NO = 30;                                   // # Of Offsets (on each side)
 query_parameter(argc, argv, qn++,              // Get # offsets
  "\n\tNumber of Positive Decoupler Offsets? ", NO);
 double offset;
 query_parameter(argc, argv, qn++,              // Get offset increment
  "\n\tDecoupler Offset Per Step (Hz)? ", offset);
 double gBsf;
 query_parameter(argc, argv, qn++,              // Get rf scaling factor
  "\n\tPulse field strength scaling factor? ", gBsf);
//              Set Up Variables Consistent Through All Offsets
 double R = (lwhh/2)*HZ2RAD;                    // Set apodization rate
 gen_op Det = Fm(sys, IsoD);                    // Set detection operator to F-
 gen_op sigma0 = sigma_eq(sys);                 // Set density mx equilibrium
 gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.);  // This is 90 detection pulse
 row_vector data(npts);                         // Block for acquisiton
//              Set Up Variables Global Over Full Profile
 row_vector profile((2*NO+1)*npts, complex0);   // Block for profile
 double totaloff = double(NO)*offset;           // Total offset at end
 row_vector fidap;                              // Block for apodized FID
 PulWaveform PWF = GP.WF();                     // GARP 25 step waveform
 PWF.scalegB1(gBsf);                            // Scale pulse waveform
 PulComposite Pcmp(PWF, sys, GP.channel());
 row_vector cyc = CYC_WALTZ4();                 // WALTZ-4 cycle overlay
 String cycname = "GARP-1 scaled";              // Modified cycle name
 PulCycle PCyc(Pcmp, cyc, cycname);             // Modified GARP-1 cycle
 SW = PCyc.FIDsync(SW);                         // Synchronize dwell times
 double td = 1/SW;                              // Set dwell time
 double tt = (npts-1)*td;                       // Total FID length
 row_vector exp=XExponential(npts,tt,0.0,R,0);  // Block for apodization
//              Loop Over Offsets, Calculate Profile
 int K =0;                                      // Point index in profile
 sys.offsetShifts(-NO*offset, IsoG);            // Set 1st profile offset
 for(int ov=-NO; ov<=NO; ov++)                  // Loop over offsets
  {
  Pcmp = PulComposite(PWF, sys, GP.channel());// Reset GARP composite pulse
  PCyc = PulCycle(Pcmp, cyc, cycname);          // Reset modified GARP-1 cycle
  data  = PCyc.FID(npts,td,Det,sigmap);         // Acquisition this offset
  fidap = product(data,exp);                    // Apodized FID this offset
  data  = FFT(fidap);                           // Spectrum this offset
  profile.put_block(0, K, data);                // Put spectrum in profile
  sys.offsetShifts(offset, IsoG);               // Move system to next offset
  K += npts;                                    // Adjust profile point index
  }
 double F = totaloff + SW/2;                    // Final plot frequency
 GP_1D("prof.asc", profile, 0, -F, F);          // Output profile ASCII data
 GP_1Dplot("prof.gnu", "prof.asc");             // Plot to screen using Gnuplot
 FM_1D("prof.mif", profile ,14,14,-F, F);       // Plot in FrameMaker MIF
 }
```

7.11.6 GARP Decoupling With Relaxation

Now lets do something a bit more exotic with GAMMA and GARP. Here we will add in the effects of relaxation while the decoupler is on. Since we already have a program that simulates GARP-1 decoupled spectra (GARPdec1.cc) without relaxation, we need only make the proper modifications to that program and its input in order to obtain the simulation we want.

Lets review a few of basic changes we'll need. First, rather than working with an isotropic spin system (spin_system) in our program, we need to work with a oriented spin system that is moving isotropically. That is, a spin system that keeps track of dipolar, CSA, and quadrupolar tensors for all spins or spin-pairs. Thus we need to replace ***spin_system*** with ***sys_dynamic***. Second, when the system is read in from an external ASCII file it will look for tensor quantities as well as dynamical values (correlation times). Next we will have to create a relaxation matrix and Liouvillian that defines how the system evolves. And lastly, we'll have to use an FID function that includes that evolves under the defined Liouvillian so that relaxation (and exchange) are accounted for.

Now that all might sound rather complicated, but it actually requires only minor adjustments to the program we already have at our disposal. Have a look. The code on the left was clipped out of the program GARPdec0.cc covered earlier in this chapter and the code on the right the modifications we need to include relaxation effects. I've left out some of the comments.

```
.                                                   .
.                                                   .
.                                                   .
spin_system sys;              // Isotropic spin system   sys_dynamic sys;              // New system type
.                                                   .
.                                                   .
.                                                   .
gen_op H = Ho(sys);           // Isotropic Hamiltonian   gen_op H = Ho(sys);           // Isotropic Hamiltonian
.                                                   super_op L = RDD(sys,H);      // Dipolar relaxation.
.                                                   .
.                                                   .
PulCycle PCyc = GP.CycGARP1(sys);                   PulCycle PCyc = GP.CycGARP1(sys, L);
.                                                   .
.                                                   .
.                                                   .
row_vector data = PCyc.FID(npts,td,Det,sigmap);     row_vector data = PCyc.FIDR(npts,td,Det,sigmap);
.                                                   .
.                                                   .
.                                                   .
```

Now that wasn't so bad was it? We generate a dipole-dipole relaxation superoperator, called "L" in the above code, and include it in the function calls which generate the GARP pulse cycle. Then we call an FID function which will include relaxation effects. Keep in mind that L resides in spin Liouville space and is typically big. Running decoupling on 5 spins will take a while and it gets worse the larger the spin system. It is a "Redfield" relaxation superoperator dealing with coupled relaxation effects (and not just longitudinal relaxation either...).

Now on to our program. The following page contains the modifications shown. I do build "L" in a different fashion than indicated above so that I can include multiple relaxation effects and exchange if I desire, rather than just setting it to only dipolar relaxation. This will become a little more clear when you look at the simulation output *versus* the input ASCII parameter file.....

7.12 Chapter Source Codes

GarpWF0.cc

```
/* GarpWF0.cc ************************************************************************  **
**                                                                                      **
**                    GAMMA GARP Simulation Example Program                             **
**                                                                                      **
** This program examines the basic GARP sequence Waveform.  I does no                  **
** NMR computations involving GARP, it merely spits out plots so that                   **
** the default GARP (GARP-1) sequence can be readily viewed.                            **
**                                                                                      **
** Assuming a.out is the executable of this program, then the following                 **
** command will generate a single 25 step GARP-1 waveform which will                   **
** be displayed on screen if Gnuplot is available.  It will also make                   **
** an editable FrameMaker MIF file of the waveform.                                     **
**                                                                                      **
**                         a.out                                                        **
**                                                                                      **
** Author:    S.A. Smith                                                                **
** Date:      2/27/98                                                                   **
** Copyright: S.A. Smith, February 1998                                                 **
**                                                                                      **
****************************************************************************************** **/

#include <gamma.h>                              // Include GAMMA

main(int argc, char* argv[])
 {
 cout << "\n\n\t\t\tGAMMA GARP Waveform Program 0\n";
 GARP GP(500.0, "1H");                          // Set GARP parameters
 PulWaveform PWF = GP.WF_GARP1();               // Construct waveform
 PWF.GP(1, 1, 5);                               // Plot waveform(s), gnuplot
 PWF.FM(1, 1, 5);                               // Plot waveform(s), Framemaker
 cout << "\n\n";                                // Keep screen nice
 }
```